**Object Detection System For Computer Mouses and Coins In A Image**

**Abstract**

In this report we will present to you our algorithm which recognizes computer mousses and coins, and count them.

Our system uses the OpenCV and NumPy libraries.

After using filters and Canny Edge Detector, we extracted contours from objects templates and compared them with the contours of the image in which we wanted to detect the objects.

In order to achieve maximum results, our pre-processing was to use image filters, morphological operators, and by choosing the correct similarity thresh-hold, we reduced false-positive and false-negative results.

**Key Words**

Image Processing, Gray Level, OpenCv2, Canny Edge Detector, Numpy, Threshold, Gaussian Opening, Morphological Operations, Object Detection.

**Methods**

Use of Filters- We used the following filters in order to get the best edge detection result

**opening** - morphological operator which combines the morphological operators Erosion which deletes "Noises" of the image and then the morphological operator Dilation which increases the line.

**Gaussian** - Reduces Noises of the image. (we found that the combination of the Opening filter and Gaussian filter gave us the best result as we will mention later)

**Sharpening Filter** - We used this filter to sharpen the image and make the edges of the objects a bit more separated from its background.

**Canny Edge Detector**: This algorithm is multi level algorithm in Image Processing, It is made for detecting edges and extracting objects from the image.

The previous filters were executed before the Canny Edge Detector in order to achieve the best results.

Since every image has different lighting, objects, noises, angels, remoteness and many more parameters that causes different results, we had to change the following thresh-holds to achieve the clearest image.

There are two thresh holds we had to determine using the Canny Edge Detector, the first threshold is for the hysteresis procedure. and the second

Contours and Detection-

**FindContours** - an openCV function that finds contours in an given image. we used this function to later compare the image's contours with a contour template of the object we wanted to detect.

**Bounding and MatchShape** - we created a function which compares contours, and once it finds two contours that match (according to the thresh hold) it draw a rectangle around the object.

The MatchShapes function's thresh-hold works in a way that the lower the result, the better is the match between the two object contours.

It is calculated based on the hu-moment values.

**Experiments in the process:**

At first we wanted to check to which filters the Canny Edge Detector will respond the best.

We wanted to use "Contrast" filter in order to make clearer differences between edges of objects and its background, by making bigger differences between each neighbor pixels.

The image after applying the "Contrast" filter and using "Canny edge detector":

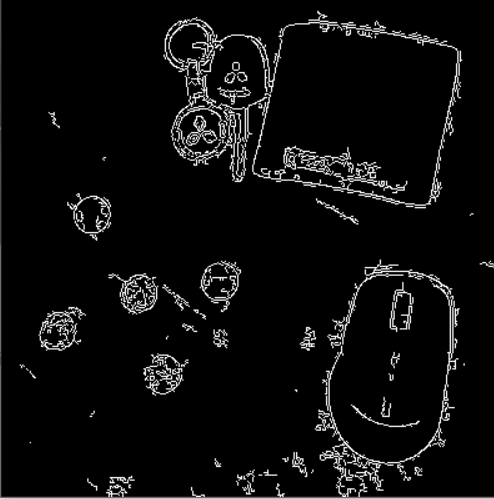


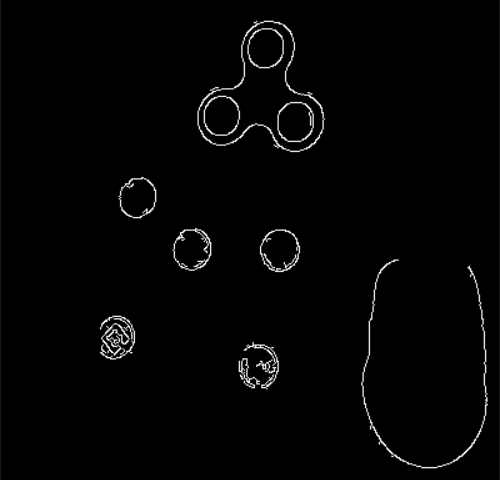
When we tested the Edge Detector we noticed that we don't get an optimal and clear result, and so we decided to try a different filter.

The image after the sharpening filter we used and "Canny edge detector":

We used sharpening filter to sharpen the image, and get better result when using the Edge Detector, but it was still full of "noises" and not clear enough.

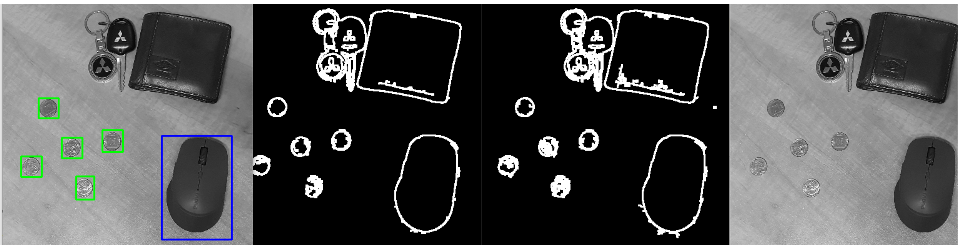
To clean these noises we used the morphological operator "Opening"

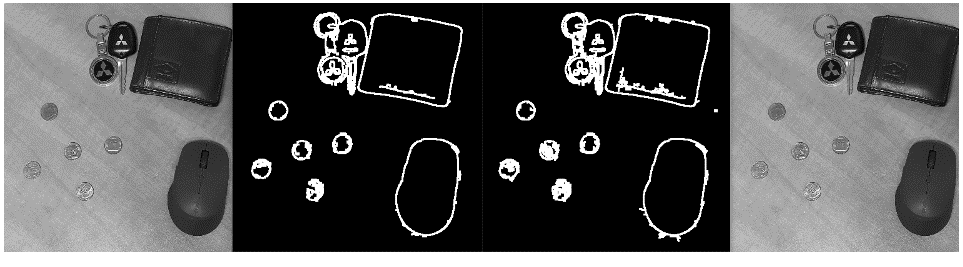
The image after "Opening" filter sharpening filter and "Canny edge detector":

 The image after "Opening" filter "Gaussian" filter sharpening filter and "Canny edge detector":

We notice that the results we got from "Canny Edge Detector" were not always as clear as needed. some of our objects have been fused with the background and there could still be some noises left in the image.

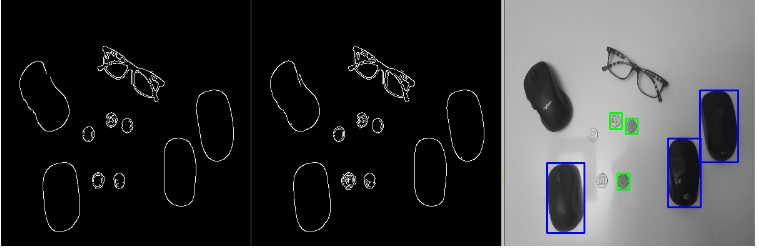
Since we used different objects, different backgrounds we had to correct the thresh-hold values on the "Canny Edge Detector".

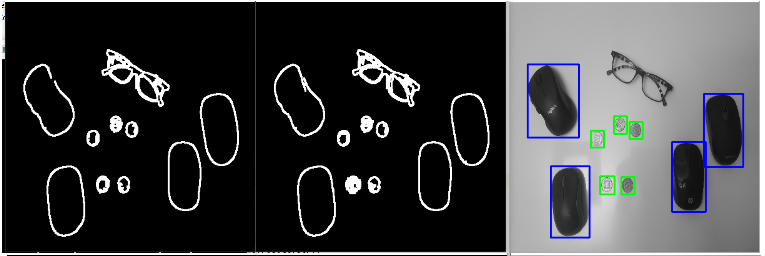
In these images you can see that when choosing the correct thresh-hold for the edge detection function, we get a full match between the contours we match.

And in this images we changed the thresh-hold to a slightly higher thresh-hold and we didn’t get any match at all.

Another problem we encountered, since we used our own images and not professional images, was that sometimes after processing the image through filters and edge detection.

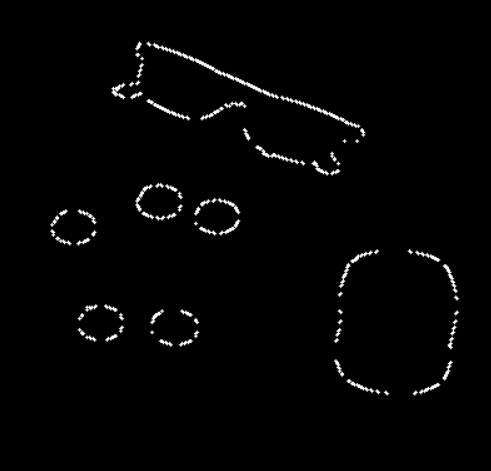
the object's contours were not entirely complete, and some of them were missed out.

In these images you can see we didn’t used dilation and the object's contours are not whole and we didn't get a full match.

and here you can see how the use of dilation makes the contours thicker and complete, and it causes a full contours match.

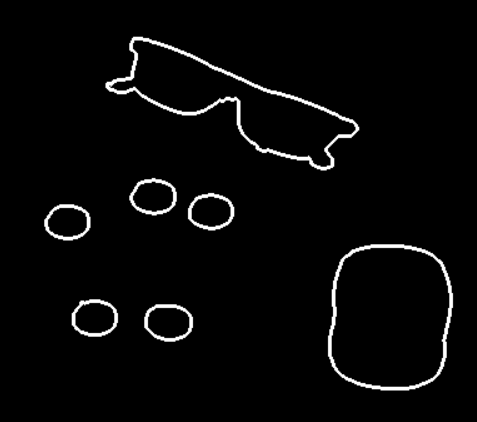
**Finding Contours-** For the purpose of detecting the contours we had to extract them in the most correct way we can, otherwise we will get matched with different object or get no results at all.

In order to understand how the function works we decided to print the contours on a black image using the NumPy library and the result was:

 1 2

As you can see in image number two, the contours we draw weren’t consecutive, and couldn’t give us good results.

To fix this we had to understand how the function works, we used the "CHAIN\_APPROX\_SIMPLE" method which dismisses some of the contours.

So we decided to use "CHAIN\_APPROX\_NONE" method, which might considered slower, since it stores the entire points of the contours, and the results were much better:

**System Description**

The purpose of the algorithm is to detect 2 different objects in an image. (we chose computer mouse and coins)

The algorithm receives three images, the first image is the image that we will the detect the objects in. the second two images are template images from which we will extract the contour of the object we want to detect

For the detection of the objects we used the MatchTemplate function which receives 2 contours and method of comparison.

**First step**

Pre-Processing**-** we used different filters to clean the image from noises and sharpen the image, and thus made the edge detection and contours extraction most efficient.

"noise cleaning" was made with morphological operator "Opening", but we also used "Gaussian" since it gave us better results.

we applied sharpening filter right after Gaussian for sharpening, and then by using Canny Edge Detector we only detected the objects in the image.

after using the edge detector with its best thresh-hold for the current image, we noticed that we are losing some lines of the objects, therefore some of the object's bordure and outline were not whole.

so we used the operator "Dilation" to get a full "frame", which made the contours much better.

Creating the Template: To get the best contour template we used a clear background with our object being in the opposite color of our background. We also tried to shot the pictures with the best lightnings we could get to get less as possible light reflection on the object or any shadows.

. By using Opening, Canny and Dilation we got the perfect contours

**Second Step-**

**Creating The Contours**: we used the function we made "create contours" to extract contours from the templates and the image.

Since this function extract many contours, we created a function ( "find\_max") which finds the contour which has the maximum area, and used it as the contour template

**Step Three-**

We created a function (create \_bounding) which receives the original image and its contours, the contour template, area size and matching thresh-hold

The function will be called separately for each object we want to detect and count how many of the specific object were found.

This function compares the contours of the original image with the contour of the template, and decides according to a given thresh-hold which of the object in the original image matches the object we wish to detect.

Once two contours match the given "matching thresh-hold", the function will add borders around the object in the original image and count it as another object which were found, hence the counting of objects.

**Results and Running Examples Detection**

**Ex1:**

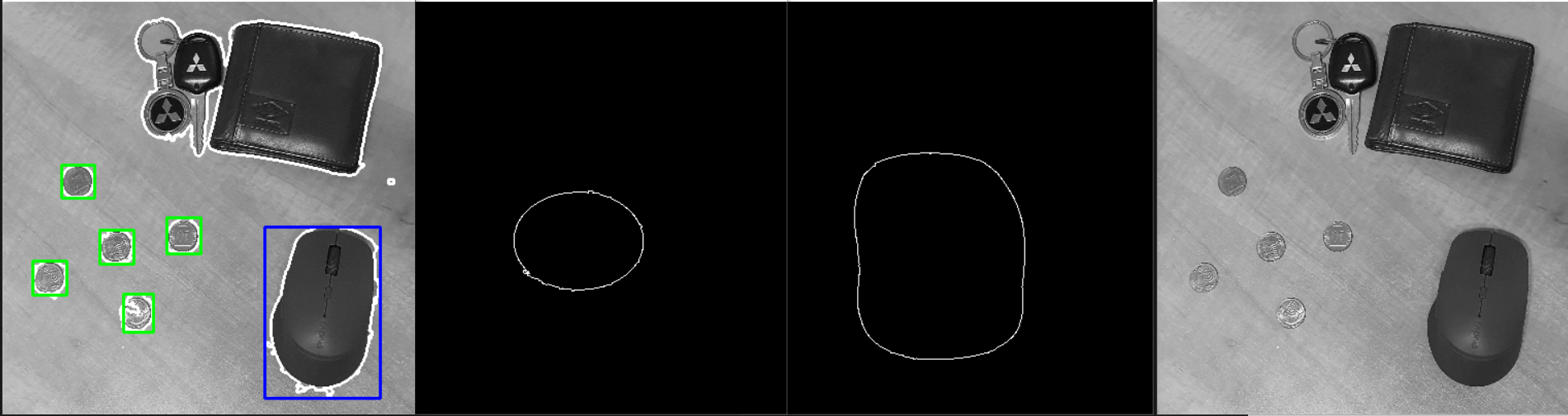
 **1 2 3 4**

Image number 1 - the original image after moving it to greyscale

Images number 2 and 3 - the contours we extracted from the templates. (computer mouse and a coin)

Image number 4 - the original image after running the algorithm. coins are marked in green and mousses are marked in blue.

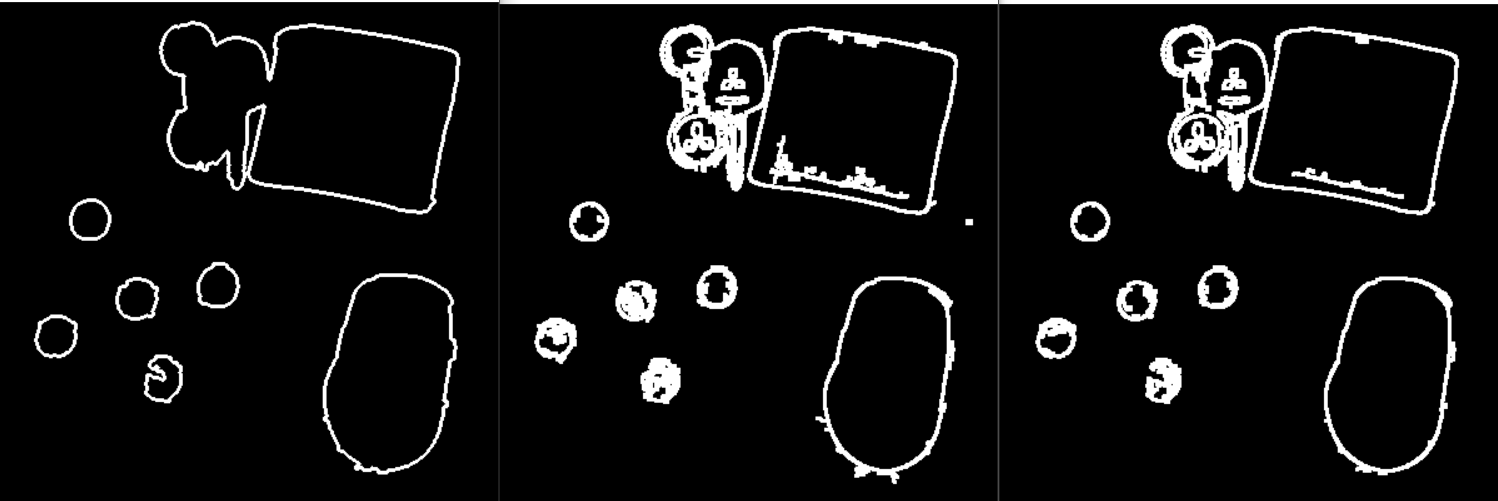
 **1 2 3**

Image number 1 and 2 - the Original image after applying the filters and edge detector.

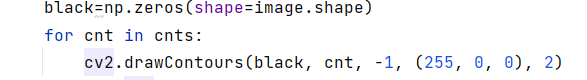
From these images we extracted the contours.

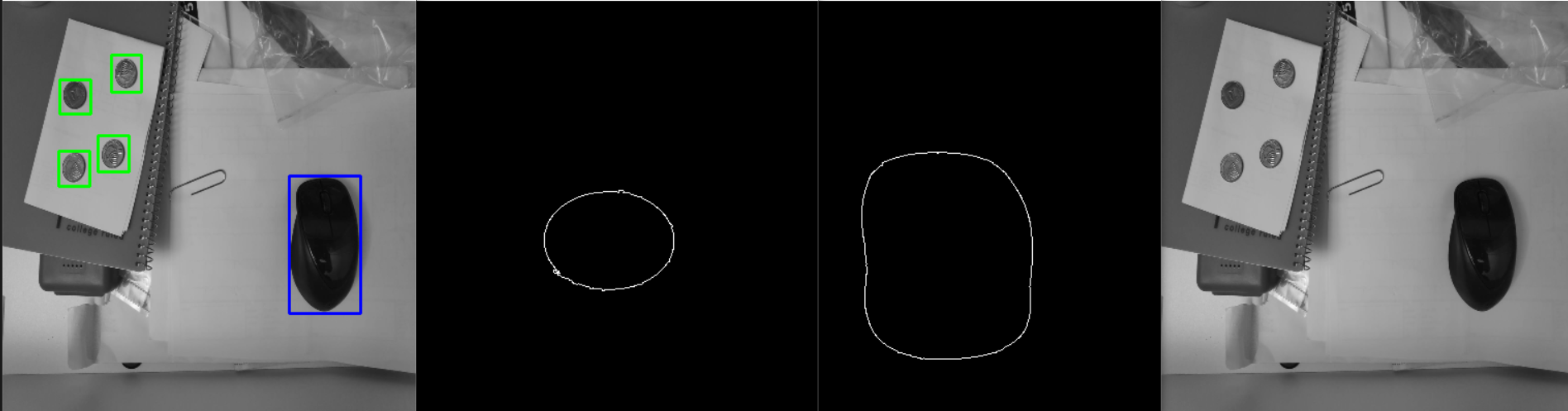
You can notice there are differences between the images. that because for each object we had to use different thresh-hold (on canny edge detector) since every object and its background act differently to each effect and thresh-hold.

Image number 3 - In this image you can see the contours of the original image after applying filters and Canny edge detector.

We used this image during writing and creating this program, in order to get a better idea of how the function "find\_contours" works, and by that we could understand if the direction we are heading towards is correct.

To create this image we used the NumPy library by filling its pixels with zeroes in order

to create a black image then printing the contours on it.

**Ex2-**

In the following images you can see a running example when we tried to use a "messier" and "noisy" background, as well as different objects of the same kind. ( different kind of mouse )

**Requirements And Limitation From The Images-**

There are various requirements and limitation the algorithm has to deal with in order to work in the perfect way

During our progress we encountered number of limitation including the following.

One of the main requirements in our project was the fact that the object has to be in a high contrast compare to its background.

It was more difficult to detect objects that their color was the same as the background.

The more similarity between the object and the background, the more difficult it was for the function "edge detection" to detect the object, since it didn't recognize the object as an edge but as a part of the background.

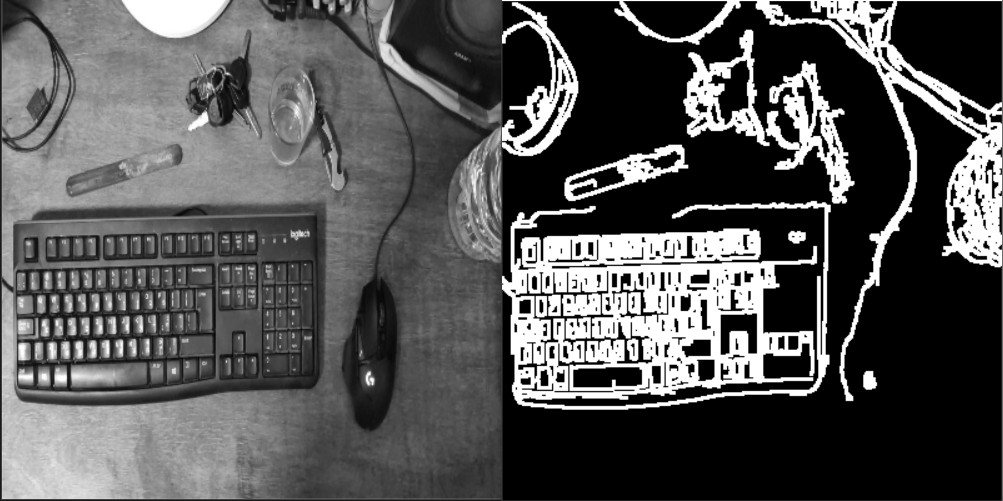
To deal with this problem, at first, we tried using "contrast" filter, in order to highlight the differences between neighbor pixels, and thus make the object a bit more separated from the background.

It didn't work as expected, it may caused a little difference between the background and the object, but it also added a lot of "noise" to the image.

We discovered that when using filters like Gaussian, and sharpening, to smoothen sharpen elements like the edges of objects, the image got clearer and we got better result.

In the next image you can see that since there was a bit of a shadow around the mouse area, the color of the mouse and part of the table around the mouse, was relatively similar.

and so, when using the Canny Edge Detector, some of the mouse lines were lost, even though we tried various thresh-holds.



Another limitation that we had is in case there were two objects close to each other, or touching one another.

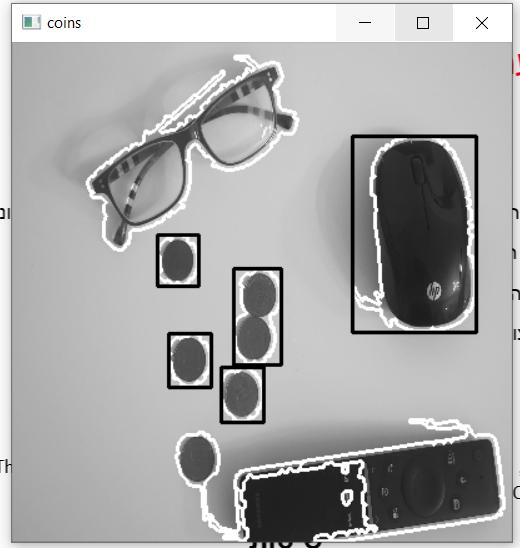
In this cases, sometimes, the object's contours got combined into one.

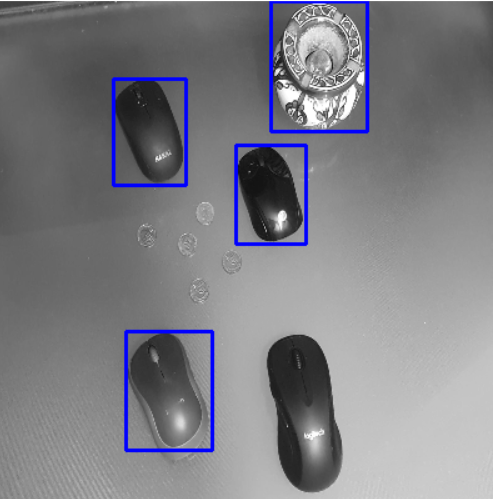
For example, when there was a shadow touching both objects, or when a bad lightning interfered the individual of each object.

When this problem occurred, we had to change some of the filters we previously used.

at first we stopped using Dilation, and then even tried to use Erosion instead.

we had many different thresh-holds inputs, but none of them were any good.

in this image you can see two coins detected as one, and another coin is not being detected due to its contour combining with the remote's contour.

**False Positive and False Negative**

In the image above you can see that while trying to detect computer mouse, we accidently detected an ashtray - False Positive.

We also didn't detect one of the mousses in the image - False Negative.

Since the contour of the ashtray has the same shape as the contour of the mouse, there was a wrong detection.

to avoid these kind of False-positive, we can change the thresh-hold of the contours comparation and run the MatchShape function with diversity of templates of the wanted object.

In this image we manage to detect most of the mouse with a relatively low thresh-hold. ( the lower the thresh-hold the bigger is the contours similarity).

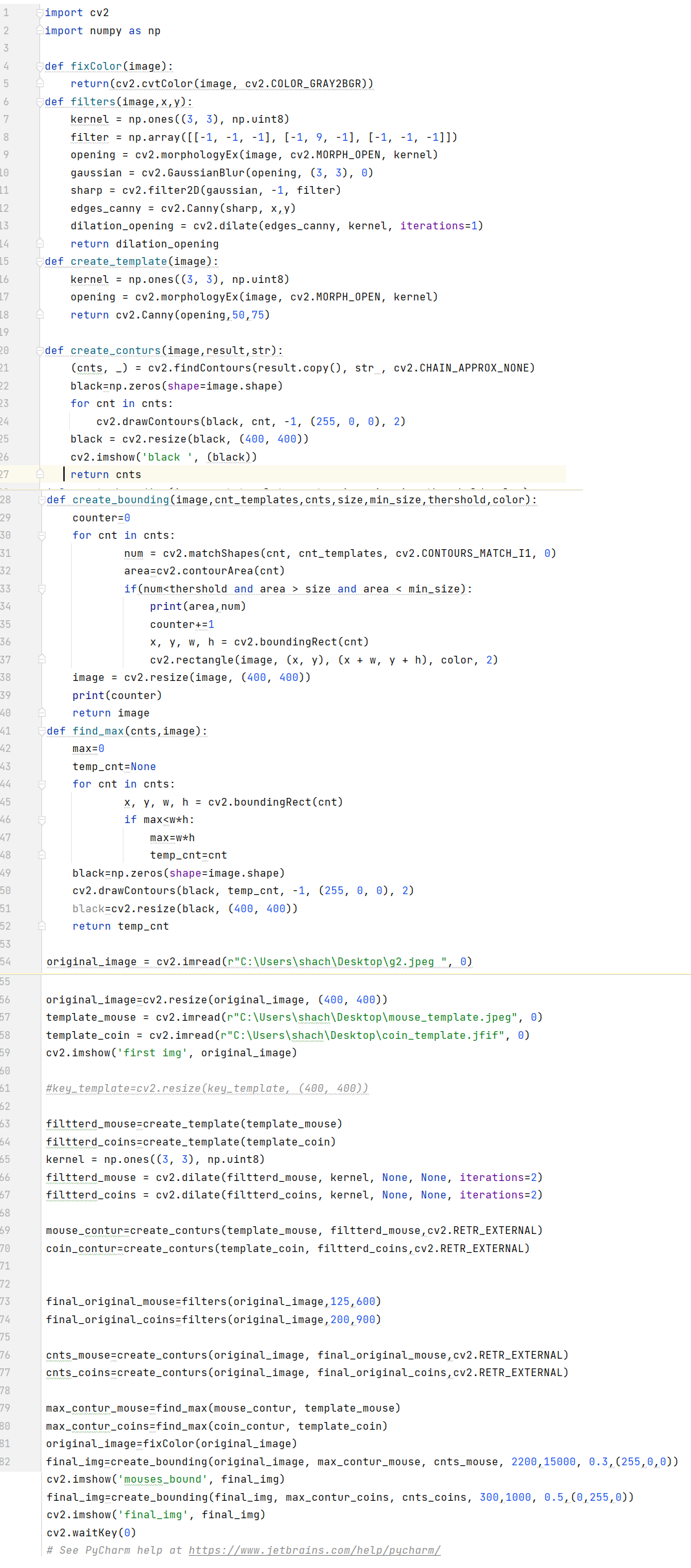


In the next image you can see 'False Positive' - the keychain is being detected as coins

in this image specifically, we fixed the problem using a different method in the function "Find\_Conturs"

In image number 1 we used: RETR\_TREE – which retrieves all of the contours and reconstructs a full hierarchy of nested contours.

In image number 2 we used: RETR\_EXTERNAL - retrieves only the extreme outer contours.



import cv2  
import numpy as np  
  
def fixColor(image):  
 return(cv2.cvtColor(image, cv2.COLOR\_GRAY2BGR))  
def filters(image,x,y):  
 kernel = np.ones((3, 3), np.uint8)  
 filter = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])  
 opening = cv2.morphologyEx(image, cv2.MORPH\_OPEN, kernel)  
 gaussian = cv2.GaussianBlur(opening, (3, 3), 0)  
 sharp = cv2.filter2D(gaussian, -1, filter)  
 edges\_canny = cv2.Canny(sharp, x,y)  
 dilation\_opening = cv2.dilate(edges\_canny, kernel, iterations=1)  
 return dilation\_opening  
def create\_template(image):  
 kernel = np.ones((3, 3), np.uint8)  
 opening = cv2.morphologyEx(image, cv2.MORPH\_OPEN, kernel)  
 return cv2.Canny(opening,50,75)  
  
def create\_conturs(image,result,str):  
 (cnts, \_) = cv2.findContours(result.copy(), str , cv2.CHAIN\_APPROX\_NONE)  
 black=np.zeros(shape=image.shape)  
 for cnt in cnts:  
 cv2.drawContours(black, cnt, -1, (255, 0, 0), 2)  
 black = cv2.resize(black, (400, 400))  
 cv2.imshow('black ', (black))  
 return cnts  
def create\_bounding(image,cnt\_templates,cnts,size,min\_size,thershold,color):  
 counter=0  
 for cnt in cnts:  
 num = cv2.matchShapes(cnt, cnt\_templates, cv2.CONTOURS\_MATCH\_I1, 0)  
 area=cv2.contourArea(cnt)  
 if(num<thershold and area > size and area < min\_size):  
 print(area,num)  
 counter+=1  
 x, y, w, h = cv2.boundingRect(cnt)  
 cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)  
 image = cv2.resize(image, (400, 400))  
 print(counter)  
 return image  
def find\_max(cnts,image):  
 max=0  
 temp\_cnt=None  
 for cnt in cnts:  
 x, y, w, h = cv2.boundingRect(cnt)  
 if max<w\*h:  
 max=w\*h  
 temp\_cnt=cnt  
 black=np.zeros(shape=image.shape)  
 cv2.drawContours(black, temp\_cnt, -1, (255, 0, 0), 2)  
 black=cv2.resize(black, (400, 400))  
 return temp\_cnt  
  
original\_image = cv2.imread(r"C:\Users\shach\Desktop\g2.jpeg ", 0)  
  
original\_image=cv2.resize(original\_image, (400, 400))  
template\_mouse = cv2.imread(r"C:\Users\shach\Desktop\mouse\_template.jpeg", 0)  
template\_coin = cv2.imread(r"C:\Users\shach\Desktop\coin\_template.jfif", 0)  
cv2.imshow('first img', original\_image)  
  
*#key\_template=cv2.resize(key\_template, (400, 400))*filtterd\_mouse=create\_template(template\_mouse)  
filtterd\_coins=create\_template(template\_coin)  
kernel = np.ones((3, 3), np.uint8)  
filtterd\_mouse = cv2.dilate(filtterd\_mouse, kernel, None, None, iterations=2)  
filtterd\_coins = cv2.dilate(filtterd\_coins, kernel, None, None, iterations=2)  
  
mouse\_contur=create\_conturs(template\_mouse, filtterd\_mouse,cv2.RETR\_EXTERNAL)  
coin\_contur=create\_conturs(template\_coin, filtterd\_coins,cv2.RETR\_EXTERNAL)  
  
  
final\_original\_mouse=filters(original\_image,125,600)  
final\_original\_coins=filters(original\_image,200,900)  
  
cnts\_mouse=create\_conturs(original\_image, final\_original\_mouse,cv2.RETR\_EXTERNAL)  
cnts\_coins=create\_conturs(original\_image, final\_original\_coins,cv2.RETR\_EXTERNAL)  
  
max\_contur\_mouse=find\_max(mouse\_contur, template\_mouse)  
max\_contur\_coins=find\_max(coin\_contur, template\_coin)  
original\_image=fixColor(original\_image)  
final\_img=create\_bounding(original\_image, max\_contur\_mouse, cnts\_mouse, 2200,15000, 0.3,(255,0,0))  
cv2.imshow('mouses\_bound', final\_img)  
final\_img=create\_bounding(final\_img, max\_contur\_coins, cnts\_coins, 300,1000, 0.5,(0,255,0))  
cv2.imshow('final\_img', final\_img)  
cv2.waitKey(0)  
*# See PyCharm help at https://www.jetbrains.com/help/pycharm/*

**Our Templates**

****

**The Images We Used**

**Special Thanks**

**We wanted to thank Dr. Cornell Lusting for a great course which give us practical and theoretical tools.**

**Assigned By:**

**Shahar Eliyahu 318254273**

**Nadav Schneider 312589021**

**Ron Hindi 318229333**